Progressive Web Apps (PWA)

What is Progressive Web Apps (PWA)?

- Modern web APIs + traditional progressive enhancement strategy = create cross-platform web applications.
- Work everywhere: desktop, tablet, and phone.
- Same user experience advantages as native apps.

Takes advantage of both web and native app features

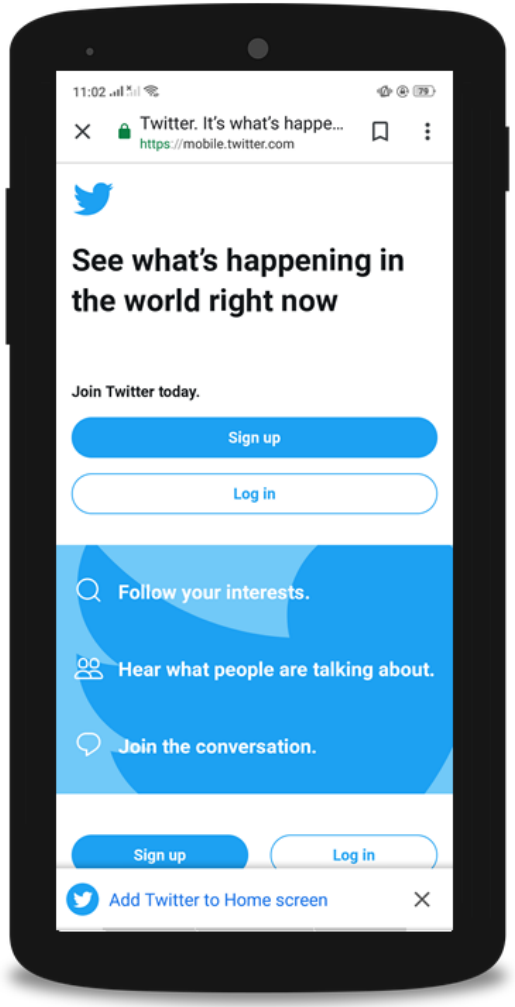
- As a web app
  - easier and faster to visit a website than install an application,
  - You can share web apps via a link.
- As a native app: can install it locally
  - Use *homescreen icons* to access the app
  - *Works offline*

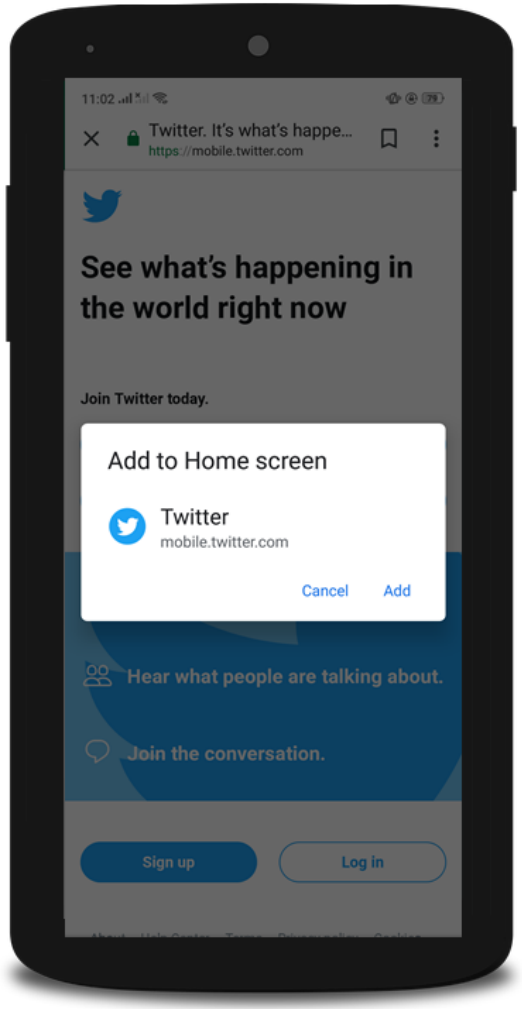
Additional features

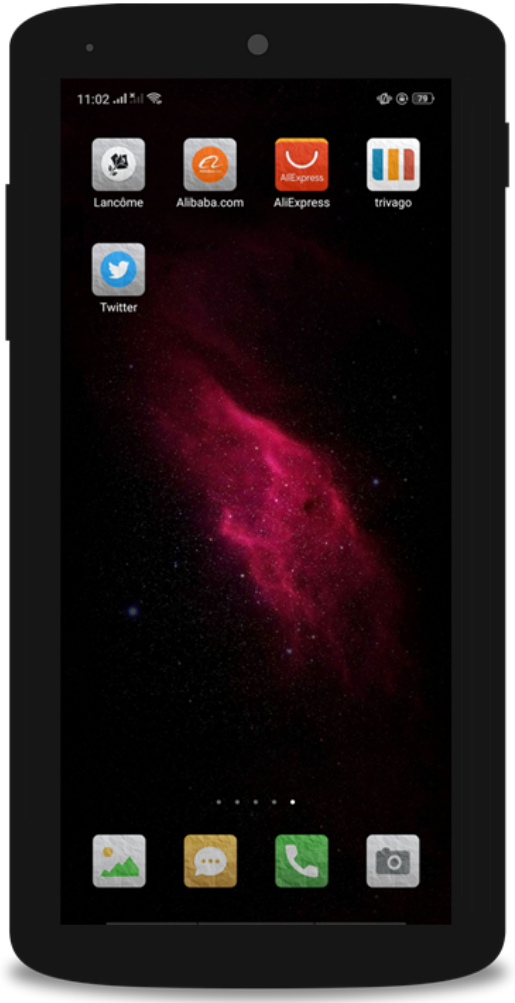
- *Discoverable*, can be found through search engines.
- *Progressive*, still usable on older browsers, and fully-functional on the latest ones.
- *Responsive*, usable on any device with a browser — phones, tablets, laptops, TVs, fridges, etc.
- *Safe*, connection between your device and app server is secured against any third parties trying to get access to your sensitive data.

Examples

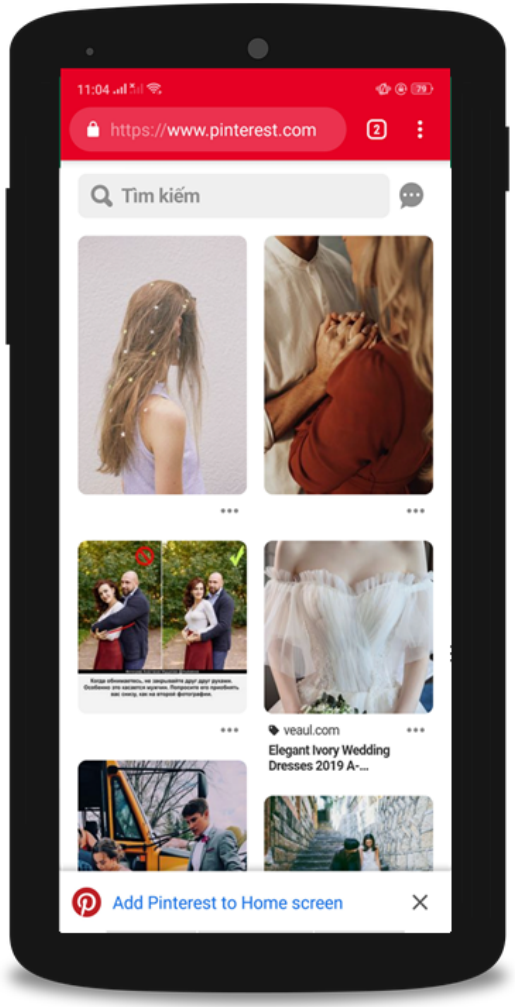
Twitter Lite

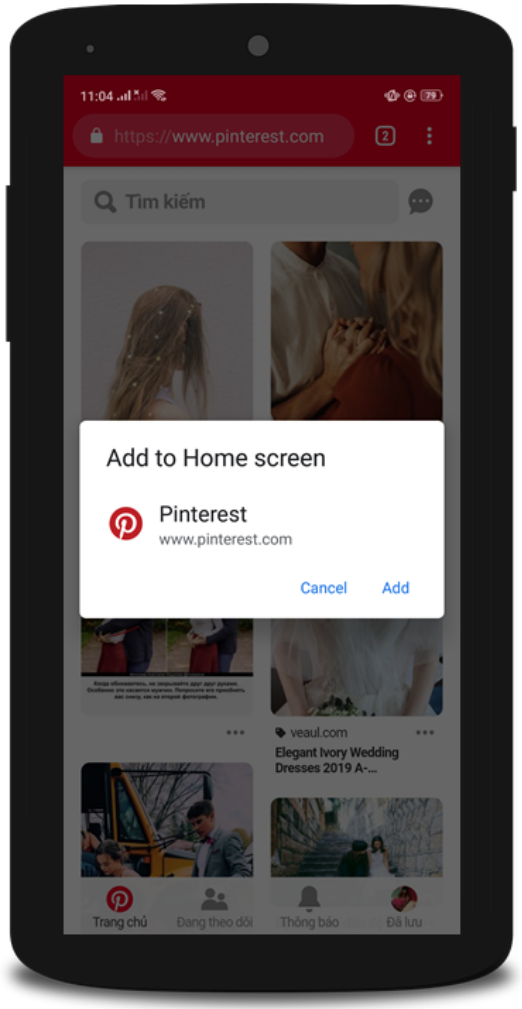


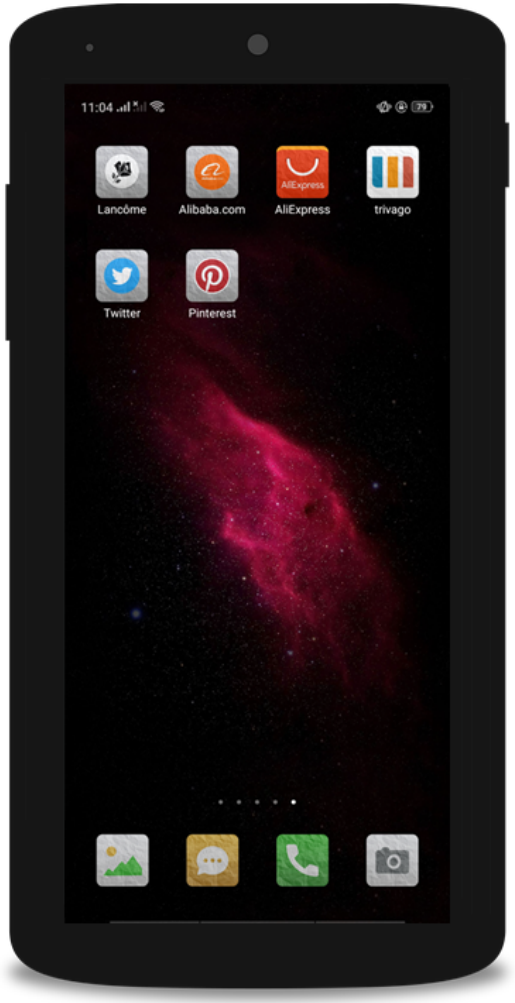




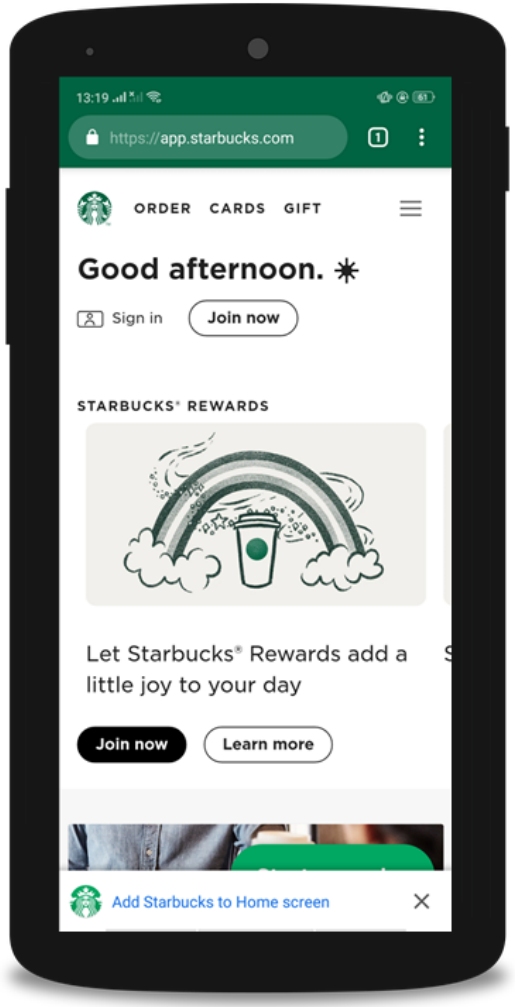
Pinterest

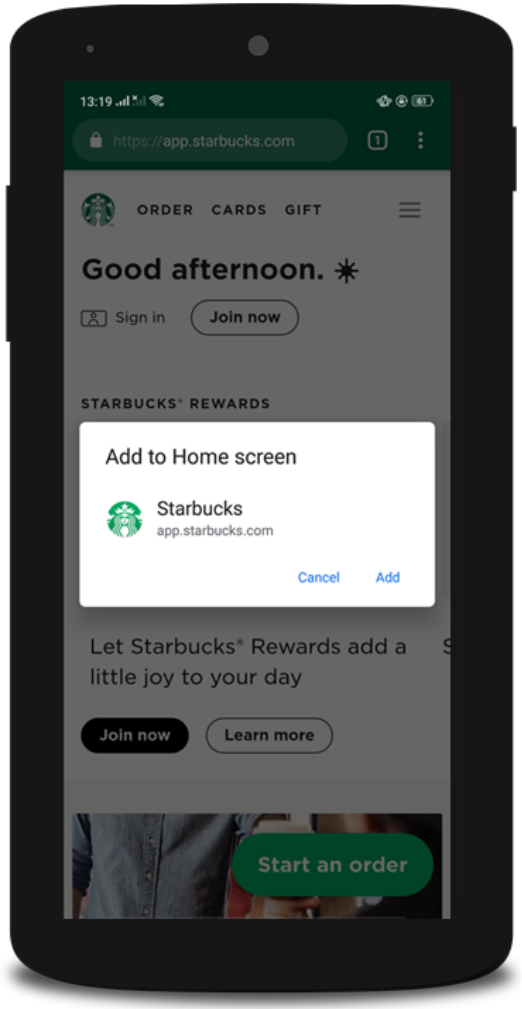

11:04
https://www.pinterest.com
Tìm kiếm
veaul.com
Elegant Ivory Wedding Dresses 2019 A-...
Add Pinterest to Home screen


11:04
https://www.pinterest.com
Tìm kiếm
Add to Home screen
Pinterest
www.pinterest.com
Cancel
Add
veaul.com
Elegant Ivory Wedding Dresses 2019 A-...
Trang chủ
Đang theo dõi
Thông báo
Đã lưu


11:04
Lancôme
Alibaba.com
AliExpress
trivago
Twitter
Pinterest

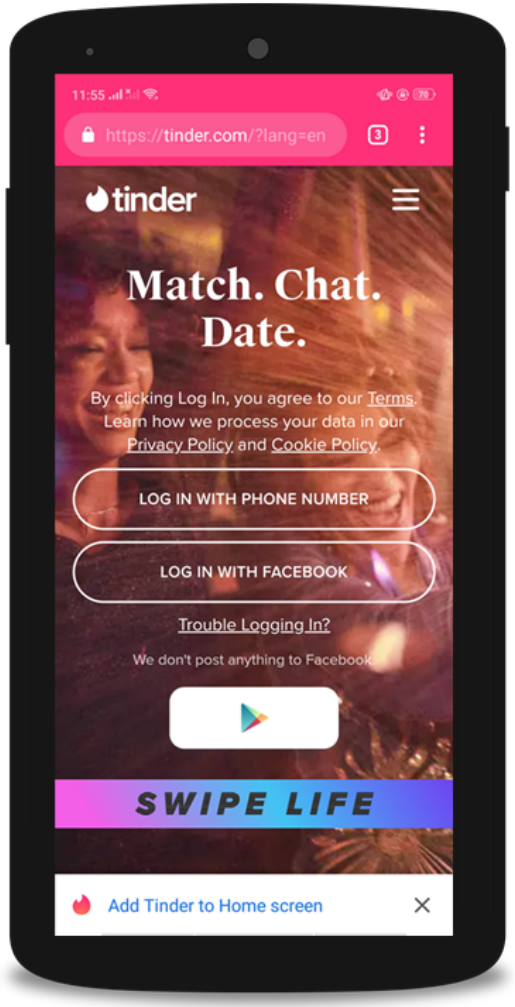
Starbucks

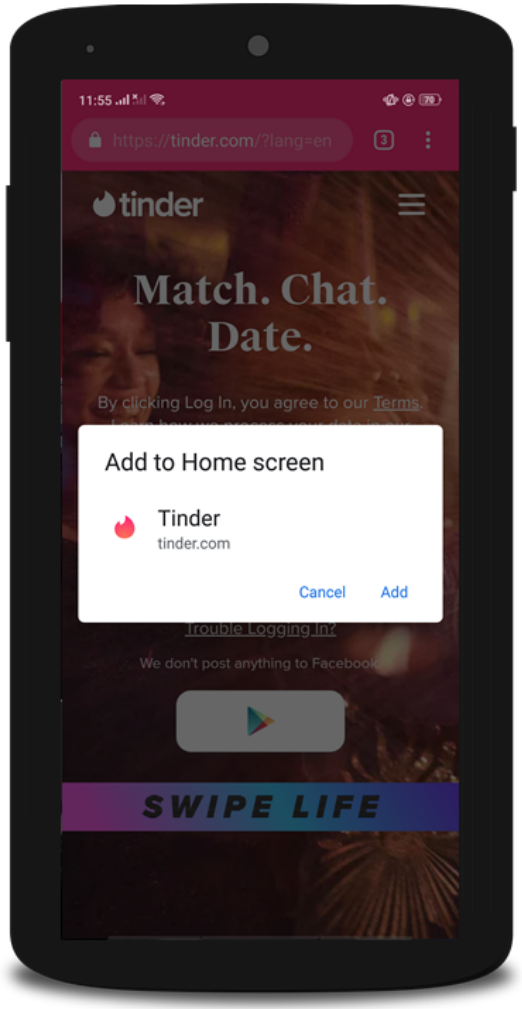

13:19
https://app.starbucks.com
ORDER CARDS GIFT
Good afternoon.
Sign in
Join now
STARBUCKS® REWARDS
Let Starbucks® Rewards add a little joy to your day
Join now
Learn more
Add Starbucks to Home screen


13:19
https://app.starbucks.com
ORDER CARDS GIFT
Good afternoon.
Sign in
Join now
STARBUCKS® REWARDS
Add to Home screen
Starbucks
app.starbucks.com
Cancel
Add
Let Starbucks® Rewards add a little joy to your day
Join now
Learn more
Start an order


13:21
Lancôme
Alibaba.com
AliExpress
trivago
Twitter
Pinterest
Jumia Travel
MakeMyTrip
PureFormulas
Flipkart Lite
Wego
OLX Lite
Magazin
nau.ch
Sound of #urbanana
LOTTO.de
Petlove
Treebo
George.com
Infobae
FT
BMW.com
NetMoms
Tinder
Telegram
Starbucks

Tinder


https://tinder.com/?lang=en
tinder
Match. Chat. Date.
By clicking Log In, you agree to our Terms. Learn how we process your data in our Privacy Policy and Cookie Policy.
LOG IN WITH PHONE NUMBER
LOG IN WITH FACEBOOK
Trouble Logging In?
We don't post anything to Facebook
SWIPE LIFE
Add Tinder to Home screen


https://tinder.com/?lang=en
tinder
Match. Chat. Date.
By clicking Log In, you agree to our Terms.
Add to Home screen
Tinder
tinder.com
Cancel
Add
SWIPE LIFE


Lancôme
Alibaba.com
AliExpress
trivago
Twitter
Pinterest
Jumia Travel
MakeMyTrip
PureFormulas
Flipkart Lite
Wego
OLX Lite
Magazin
nau.ch
Sound of #urbanana
LOTTO.de
Petlove
Treebo
George.com
Infobae
FT
BMW.com
NetMoms
Tinder

Example: pet store

- We will continue with the 'pet store' example we had earlier.
- Turn it into a PWA
  - can install locally
  - add a shortcut to the home screen
  - work offline

The Pet Store - the `<head>`

```
<html>
<head>
    <title>Vue.js Pet Depot</title>
    <script src="https://unpkg.com/vue"></script>
    <script src="products.js"></script>
    ...
</head>
<body>
    ...
```

The product list - `products.js`

```
let products = [
    {
      "id": 1001,
      "title": "Cat Food, 25lb bag",
      "description": "A 25 pound bag of <em>irresistible</em>, ...",
      "price": 20.00,
      "image": "images/cat-food.png",
      "availableInventory": 10,
      "rating": 2
    },
    {
      "id": 1002,
      "title": "Yarn",
      "description": "Yarn your cat can play with for a very ...",
      "price": 2.99,
      "image": "images/yarn.jpg",
      "availableInventory": 7,
      "rating": 3
    },
    ...
  ]
```

The Pet Store - the Vue.js

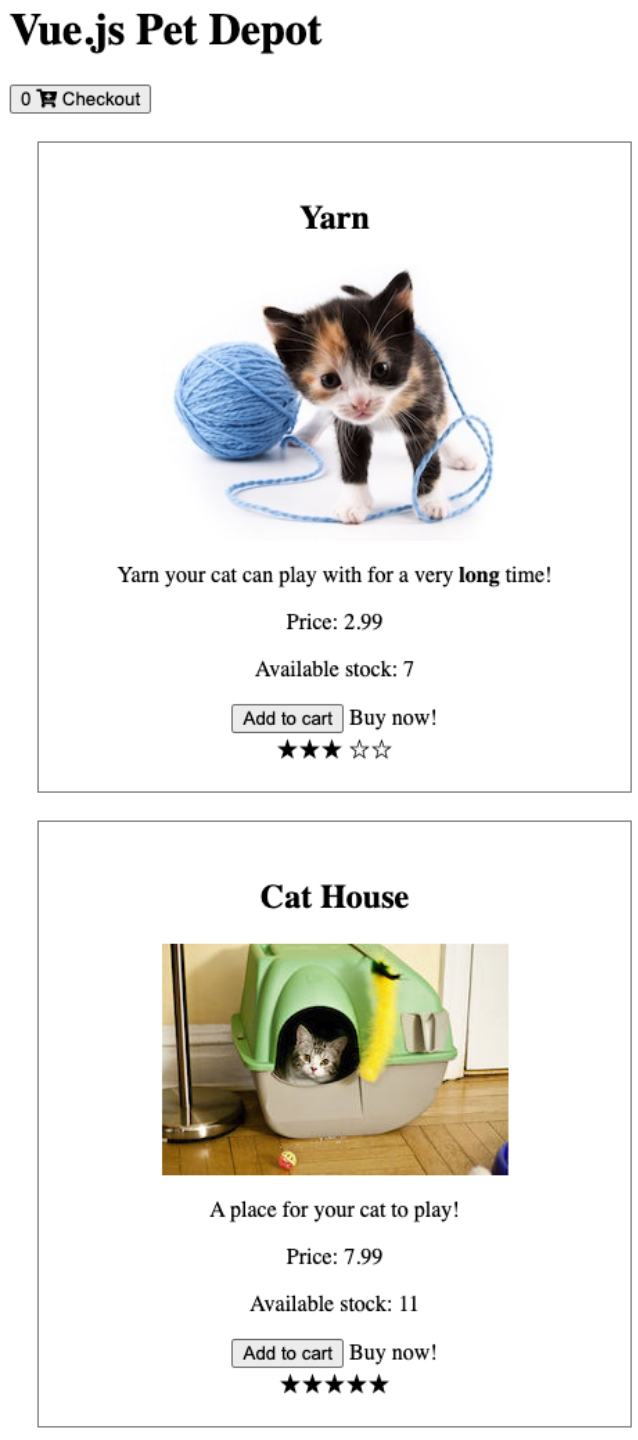
```
var webstore = new Vue({
    el: '#app',
    data: {
        products: products,
        cart: [],
        ...
    },
    methods: {
        addToCart(product) {
            this.cart.push(product.id);
        },
        ...
    },
    computed: {
        sortedProducts() {...}
        ...
    },
});
```

The Pet Store - the products

```
<div id="app">
    ...
    <div v-for="product in sortedProducts">
        <h2 v-text="product.title"></h2>
        <figure>
            <img v-bind:src="product.image">
        </figure>
        <p v-html="product.description"></p>
        <p>Price: {{product.price}}</p>
        <p>Available stock: {{product.availableInventory}}</p>
        <button v-on:click='addToCart(product)'
            v-if='canAddToCart(product)'>
            Add to cart
        </button>
        ...
    </div>
    ...
</div>
```

Tidy up the formatting

```
.product {
    border: 1px grey solid;
    padding: 20px;
    margin: 20px;
    text-align: center;
}
```



Make PWAs installable

- A web manifest, with the correct fields filled in
- The website to be served from a secure (HTTPS) domain
    - We will use GitHub Pages, which provide HTTPS connection
- An icon to represent the app on the device
- A service worker registered, to make the app work offline

The manifest file

- Lists all the information about the PWA in a JSON format.
- It usually resides in the root folder of a web app.
- It contains useful information, such as
  - the app's name,
  - paths to different-sized icons (for example, as the home screen icon)
  - a background color to use in loading or splash screens.
- The manifest file has an extension of `.webmanifest` (`.json` also works).
- Link the manifest file in the `<head>` section of the html file:

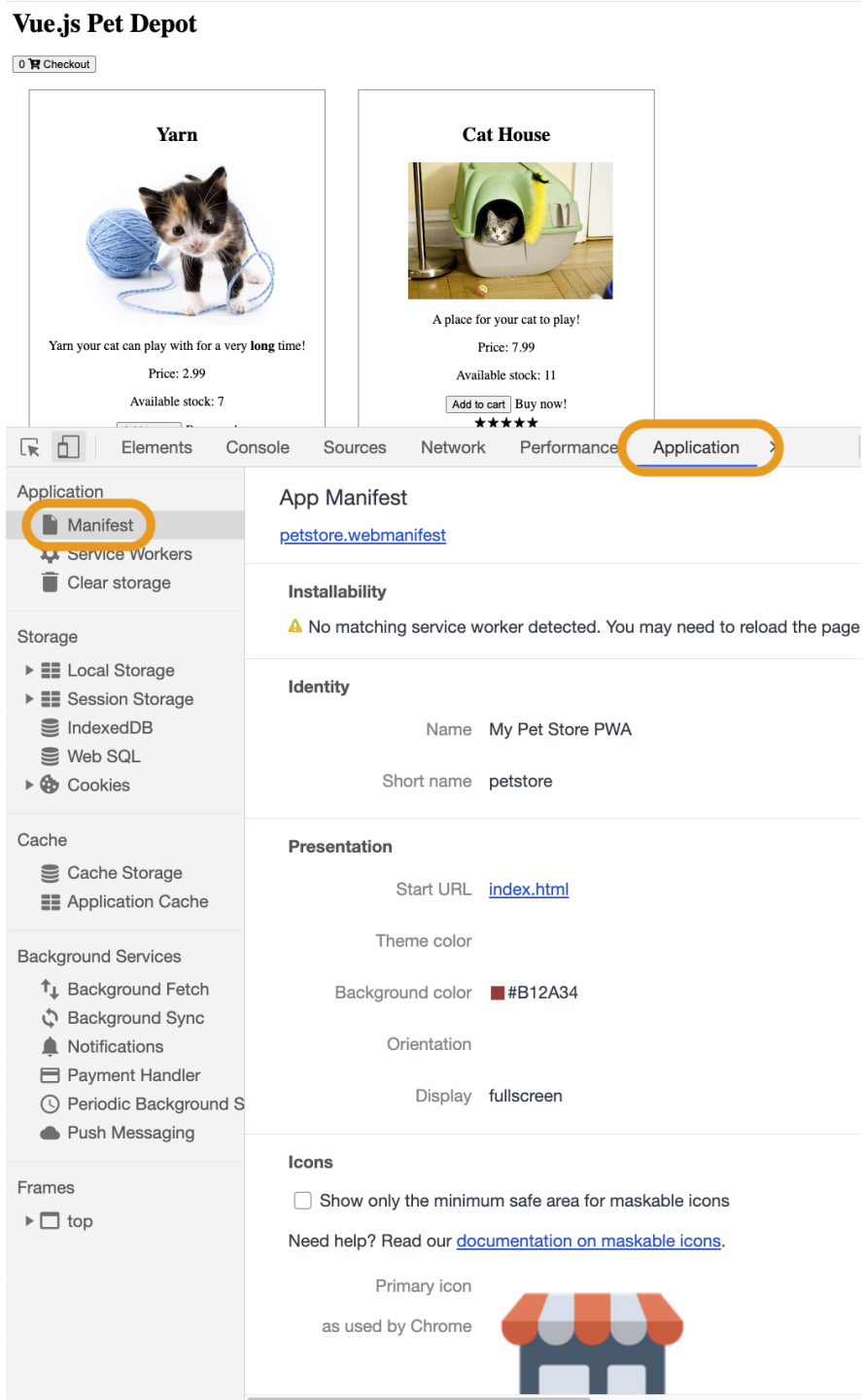
```
<link rel="manifest" href="petstore.webmanifest">
```

petstore.webmanifest

```
{
    "name": "My Pet Store PWA",
    "short_name": "petstore",
    "description": "my online pet store",
    "icons": [
        {
            "src": "images/icon-512.png",
            "sizes": "512x512",
            "type": "image/png"
        }
    ],
    "start_url": "index.html",
    "display": "fullscreen",
    "background_color": "#B12A34"
}
```

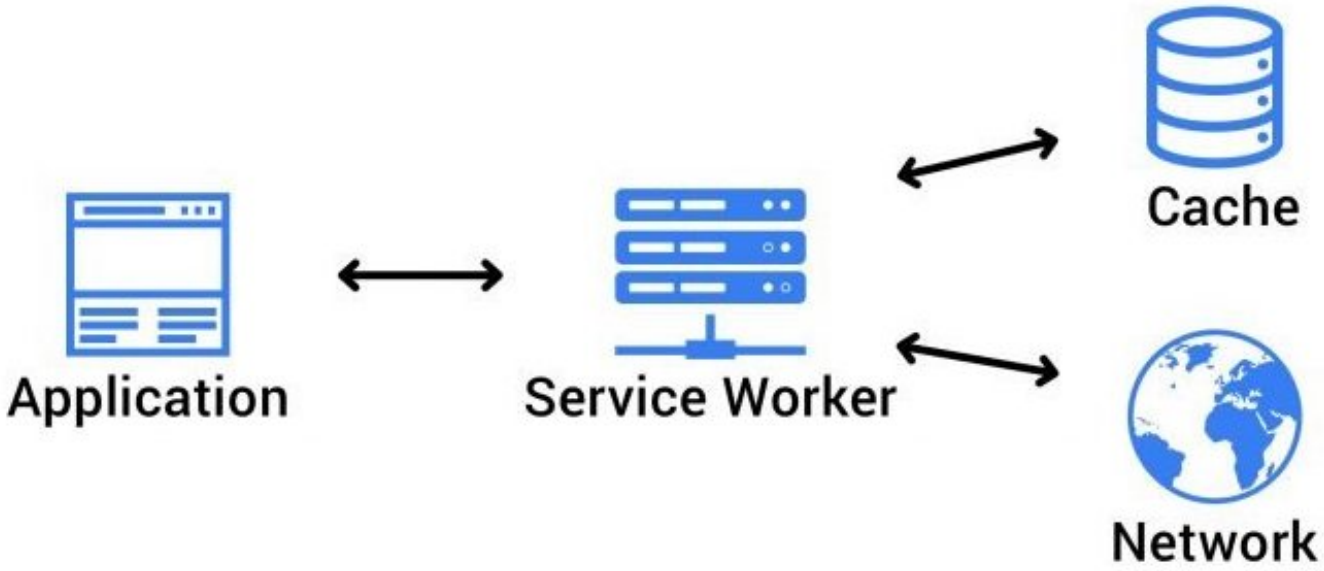
Open the app in browser

- It seems nothing has changed if you open the app in the browser now.
- If you go to the browser devtools >> 'Application' >> 'Manifest'
  - You can see Chrome recognised the `pestore.webmanifest` file
  - Showing values such as 'name', 'start URL', and 'background color'
- However, there is still an error 'Installability: No matching service worker detected'.



Service workers

- Service Workers are a *virtual proxy* between the browser and the network.
- It can *cache* the files of a website and make them available when the device is offline.



- They run on *a separate thread* from the main JavaScript code
  - Don't have any access to the DOM structure.
- The API is *non-blocking*.
  - You can give a Service Worker something to work on, and receive the result when it is ready using a *Promise*-based approach.
- Service Workers can only be executed in secure contexts (meaning HTTPS)

Registering a Service Worker (in the main html file)

```
if('serviceWorker' in navigator) {
    navigator.serviceWorker.register('service-worker.js');
};
```

- If the service worker API is supported in the browser, it is registered using the `navigator.serviceWorker.register()` method.
- Its contents reside in the `service-worker.js` file, and can be executed after the registration is successful.
- When registration is complete, the `service-worker.js` file is automatically downloaded, then installed, and finally activated.

Caching the files (in `service-worker.js`)

- First, a variable for storing the cache name is created,
- Then, the files to be cached are listed in an array.

```
var cacheName = 'petstore-v1';
var cacheFiles = [
  'index.html',
  'product.js',
  'petstore.webmanifest',
  'images/yarn.jpg',
  'images/cat-litter.jpg',
  'images/laser-pointer.jpg',
  'images/cat-house.jpg',
  'images/icon-store-512.png'
  ...
];
```

Caching the files (in `service-worker.js`)

- Then we can cache the files in the 'install' event listener:

```
self.addEventListener('install', (e) => {
  console.log('[Service Worker] Install');
  e.waitUntil(
    caches.open(cacheName).then((cache) => {
      console.log('[Service Worker] Caching all the files');
      return cache.addAll(cacheFiles);
    })
  );
});
```

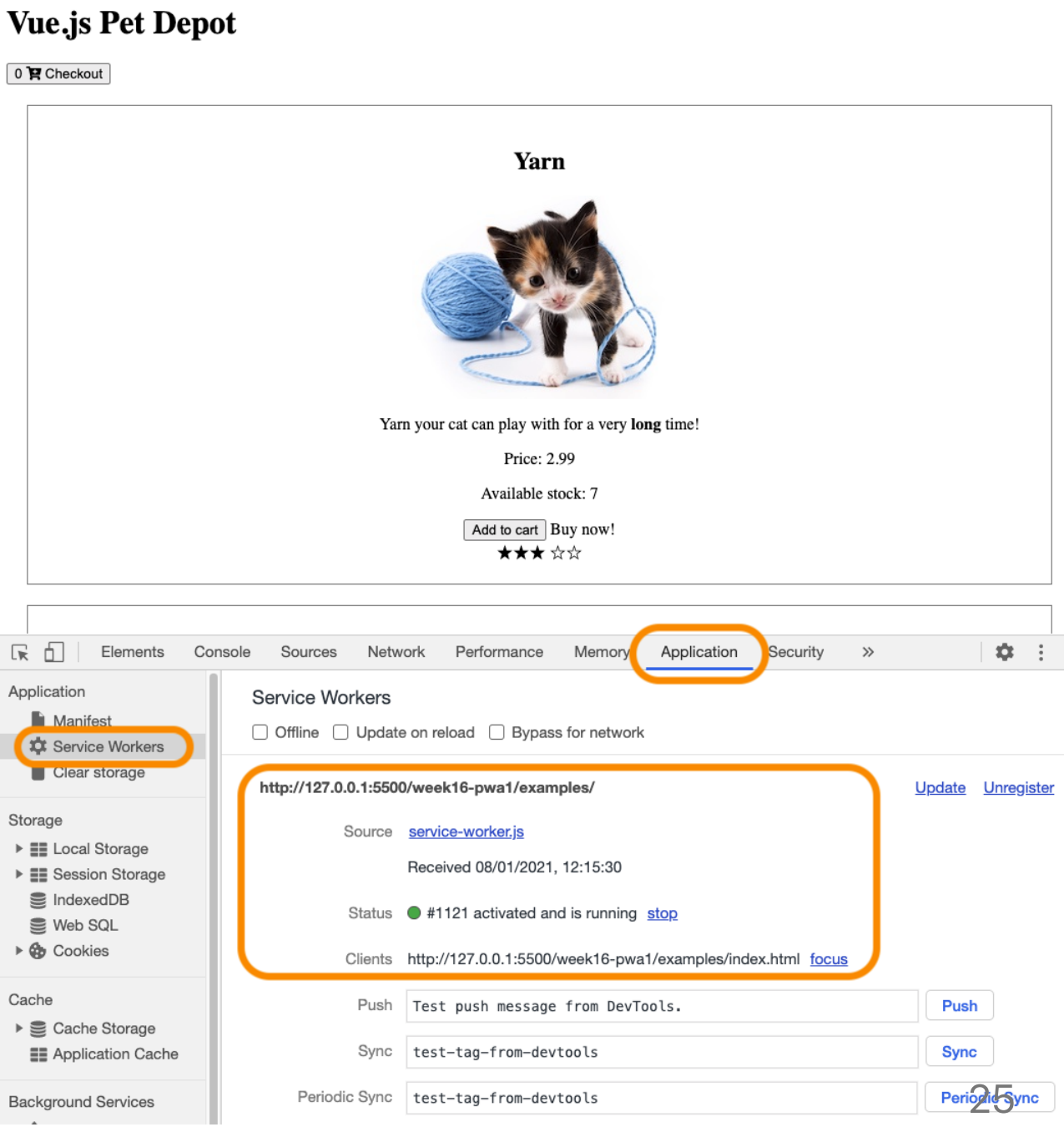
- The `self` here refers to the `window` object in JavaScript (the browser window).
- The service worker does not install until the code inside `waitUntil` is executed.
    - It returns a promise

```
self.addEventListener('install', (e) => {
  console.log('[Service Worker] Install');
  e.waitUntil(
    caches.open(cacheName).then((cache) => {
      console.log('[Service Worker] Caching all the files');
      return cache.addAll(cacheFiles);
    })
  );
});
```

- The `caches` is a special 'Cache Storage' object for Service Worker to save data
  - Similar to 'local storage', but for caching only
- Here, we open a cache with a given name (`petstore-v1`),
- Then add all the files (listed in `cacheFiles`) to the cache,
  - So they are available next time it loads.

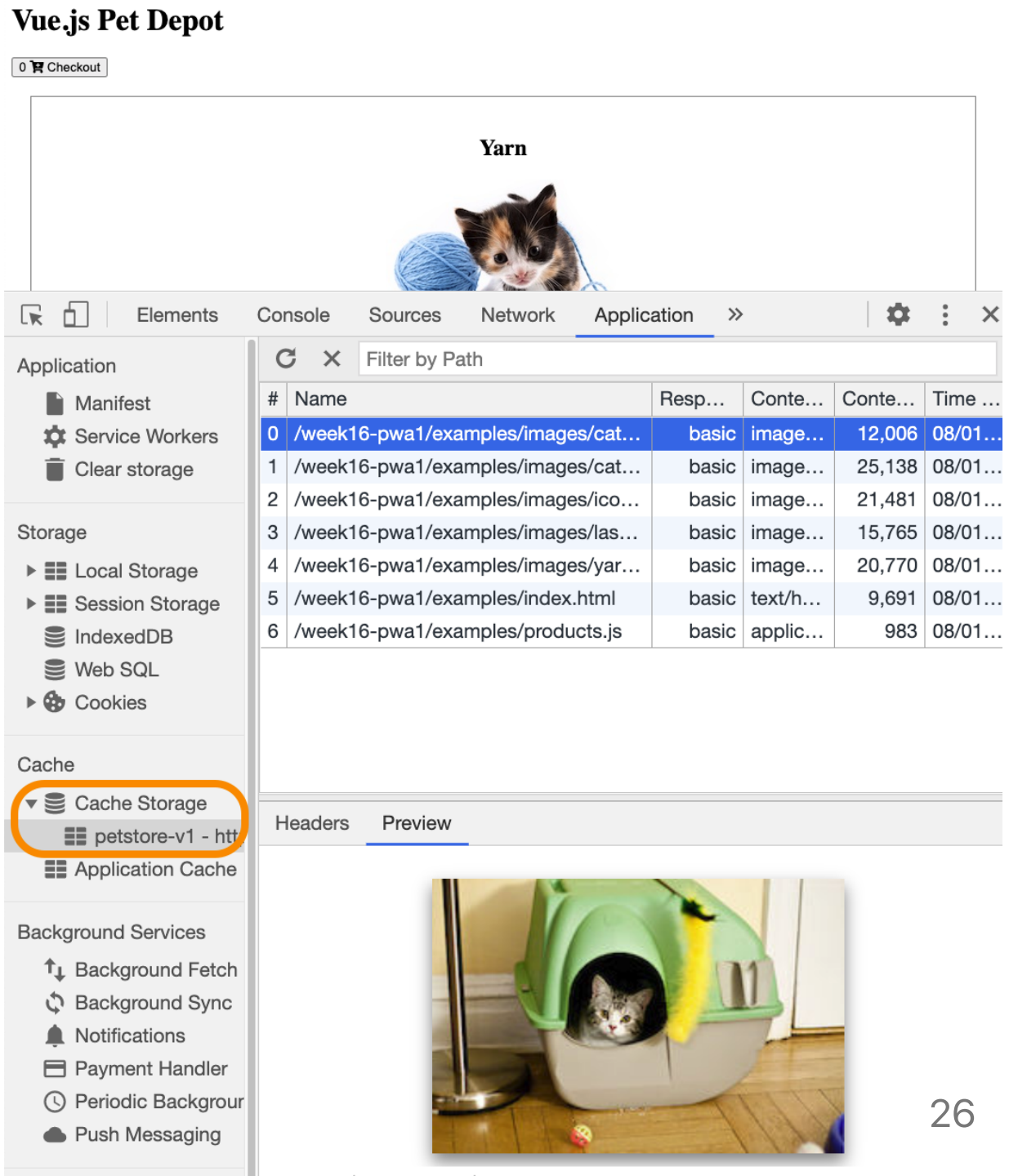
Run the service worker

- If you open the app again in the browser, the page looks the same
- If you go to devtools >> Application >> Service Workers
- It should show that `service-worker.js` is activated and running
- If there is any error, make sure the file names in the list are correct.



The cached files

- If you select 'Cache Storage' from the left pane, there should a cache named `petstore-v1`
- Clicking on `petstore-v1` will show the list of files that have been added to the cache in the top right pane
- Clicking on any of the file will show a preview in the bottom right pane



Using the cached files

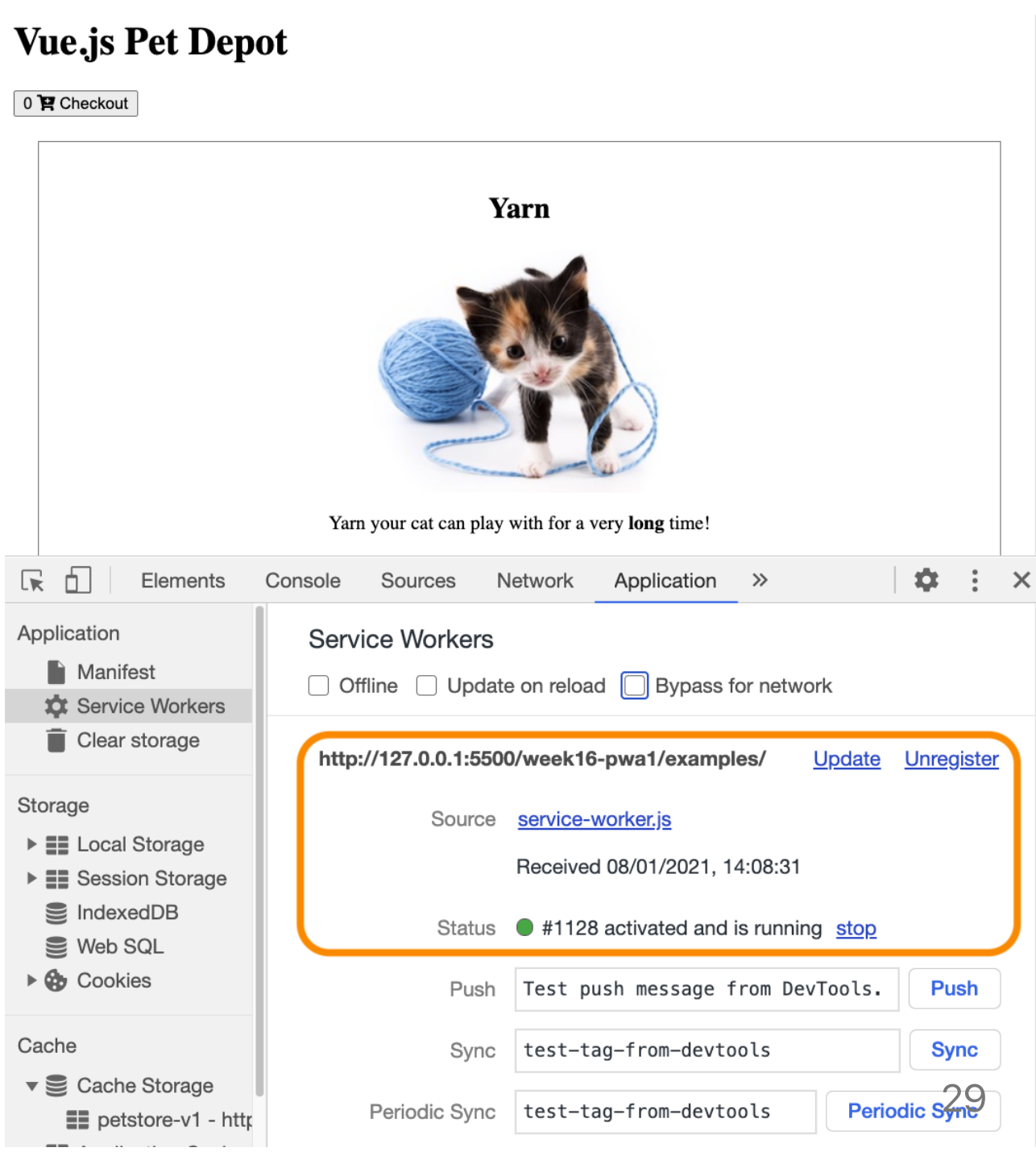
- Now that the files are cached, we can use the local files when starting the app rather than retrieving them from the server
    - This should be faster and also allow the app to work offline
- The ways to get this to work is to 'intercept' any `fetch` request
    - When the front end sends any `fetch` request to get data from the server
    - The service worker will redirect the request to the cache and return the file there
- This is achieved by listening to the `fetch` event:

```
self.addEventListener('fetch', (e) => {
    console.log('[Service Worker] Fetched resource '+e.request.url);
});
```

```
self.addEventListener('fetch', function (e) {
    e.respondWith(
        // check if the cache has the file
        caches.match(e.request).then(function (r) {
            console.log('[Service Worker] Fetching resource: '
              + e.request.url);
            // 'r' is the matching file if it exists in the cache
            return r
        })
    );
});
```

- Here, we respond to the fetch event with a function that tries to find the resource in the cache and return the response if it's there.
- The `FetchEvent.respondWith` method intercepts all `fetch` request
    - functions as a proxy server between the app and the network.

Now the service worker will look all fine in the devtools



However, there are still a few errors if you refresh the page

- When refreshed, all the files are loaded from the cache
  - Because we intercept every fetch request
- While html and images are there
  - Third-party files are missing
  - Such as the Vue.js and css for FontAwesome

Caching new files

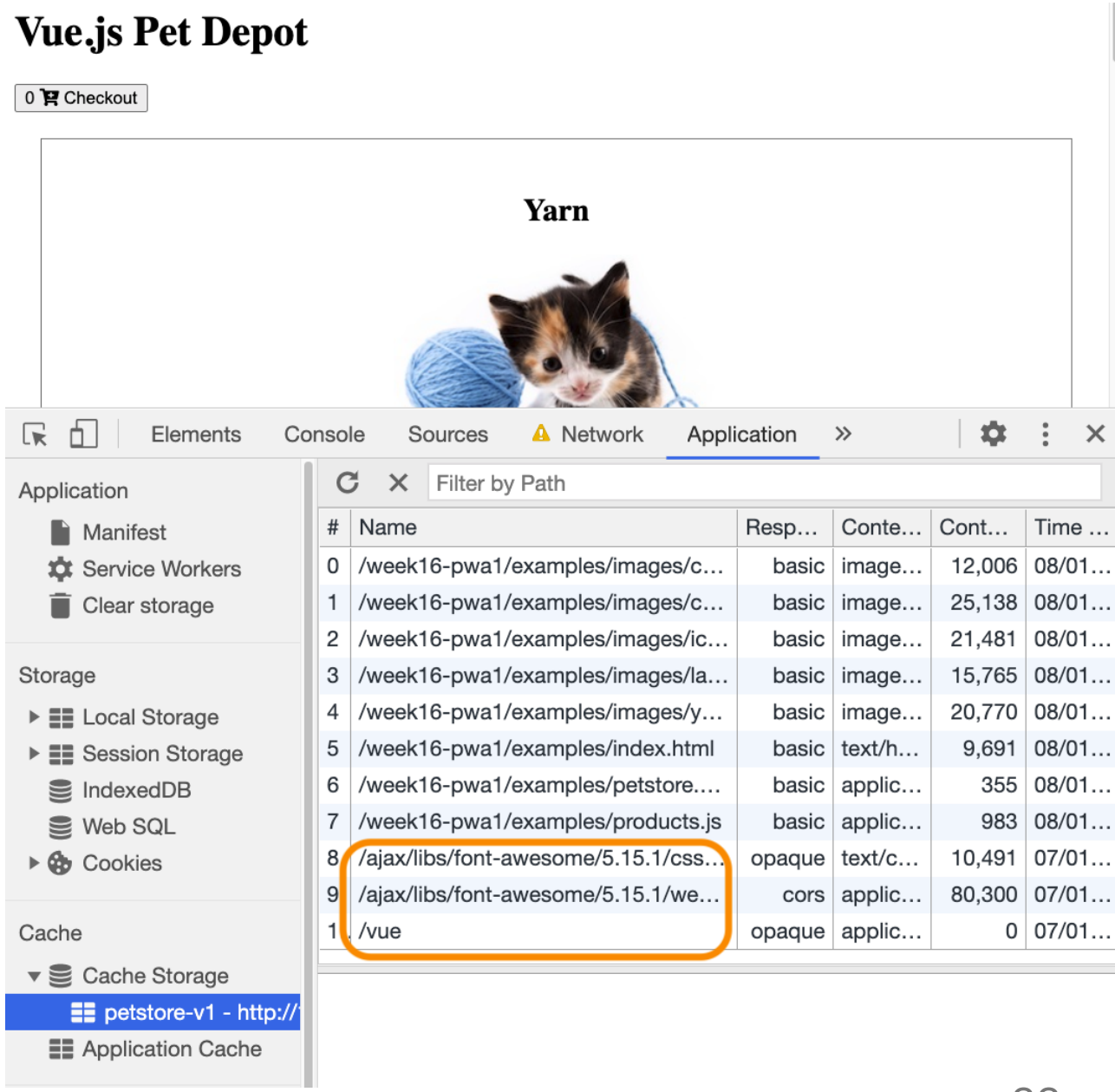
- While it is possible to add all the missing files to the caching file list
- The alternative is to download missing files dynamically and add them to cache

```
self.addEventListener('fetch', function (e) {
  e.respondWith(
    caches.match(e.request).then(function (r) {
      // Download the file if it is not in the cache,
      return r || fetch(e.request).then(function (response) {
        // add the new file to cache
        return caches.open(cacheName).then(function (cache) {
          cache.put(e.request, response.clone());
          return response;
        });
      });
    })
  );
});
```

- `fetch(e.request)`: if the file is not in the cache, we use another fetch request to download it,
- `caches.open(cacheName).then(...)`: then store the response in the cache so it will be available there next time it is requested.
- This allows us to respond to every single request with any response we want:
    - This includes other request type besides `GET`
    - The service worker have full control of the response (and can be potentailly used for malicious purpose)

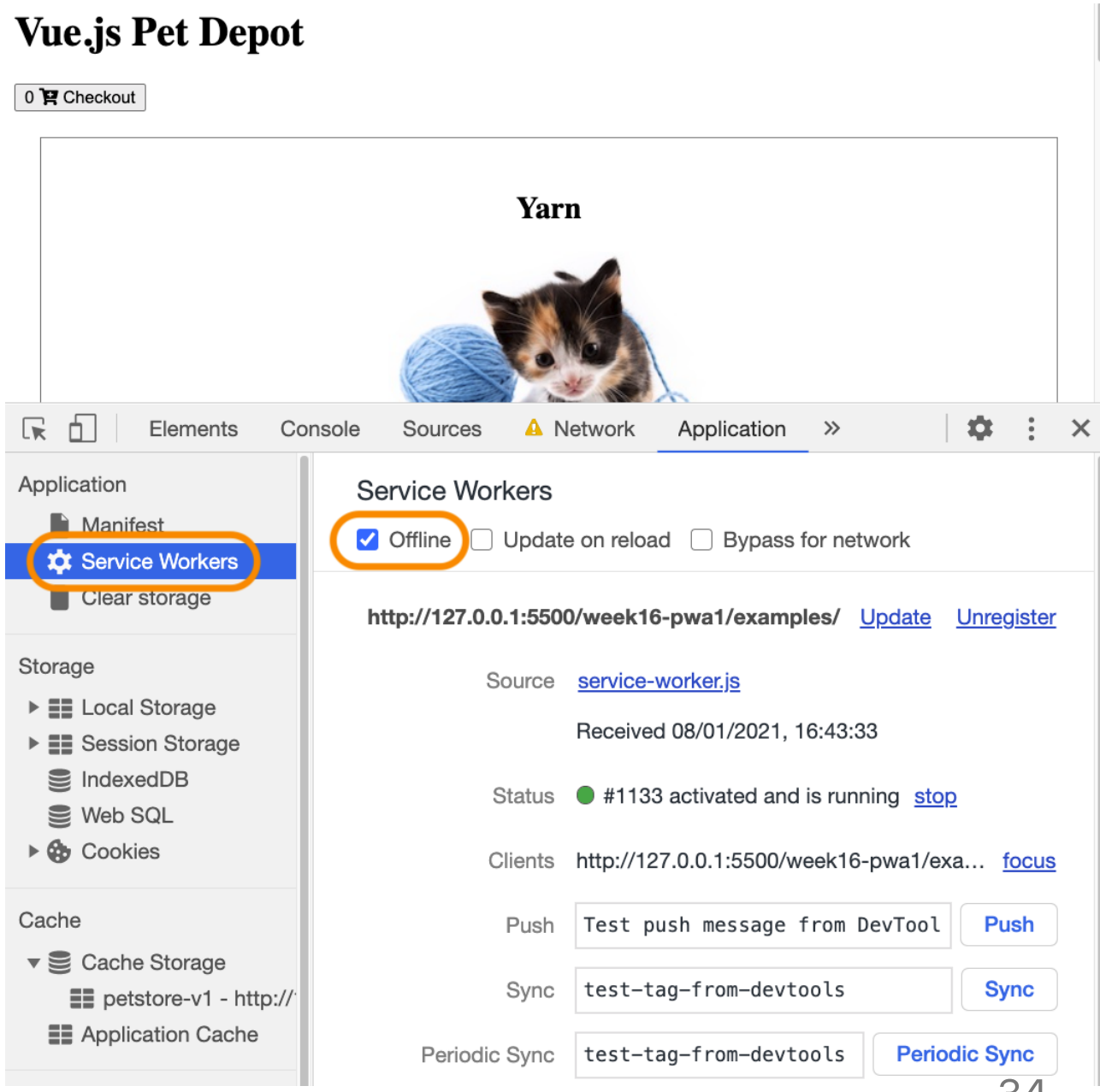
Now the missing files are added to the cache when the page reloaded

- The files for FontAwesome and Vue are now added to the cache



You can also test this in the 'Service Workers' pane with the 'Offline' option

- This disconnects the network
- And the app should still work fine after refresh

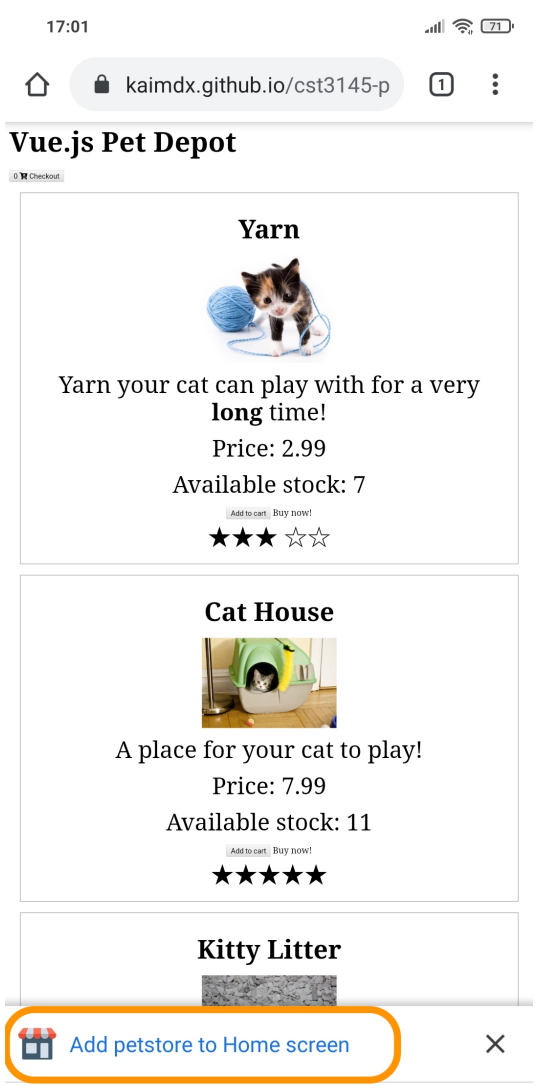


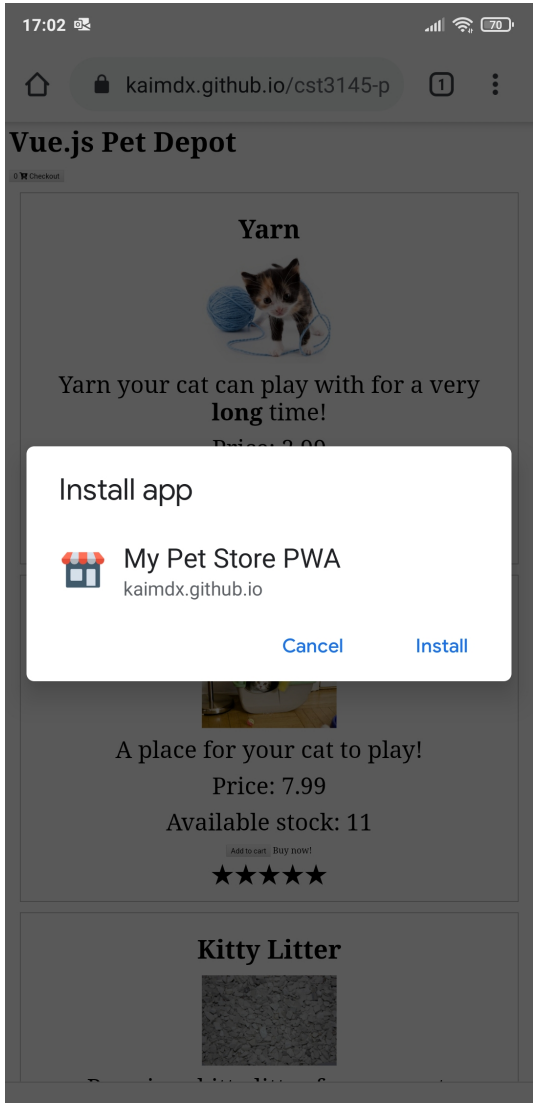
Now we meet all the requirements to make PWA installable

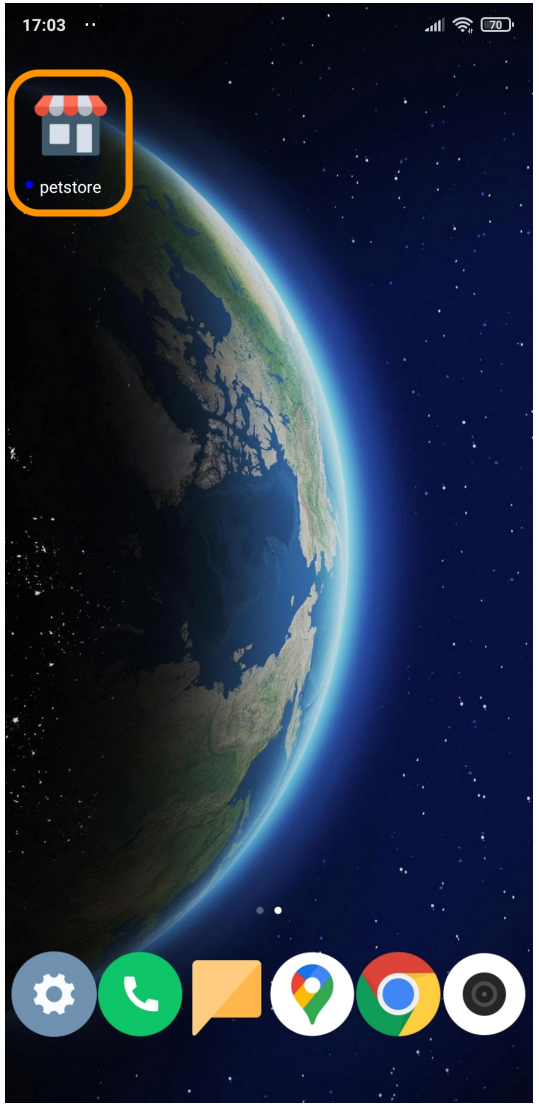
Just need to upload it to GitHub Pages

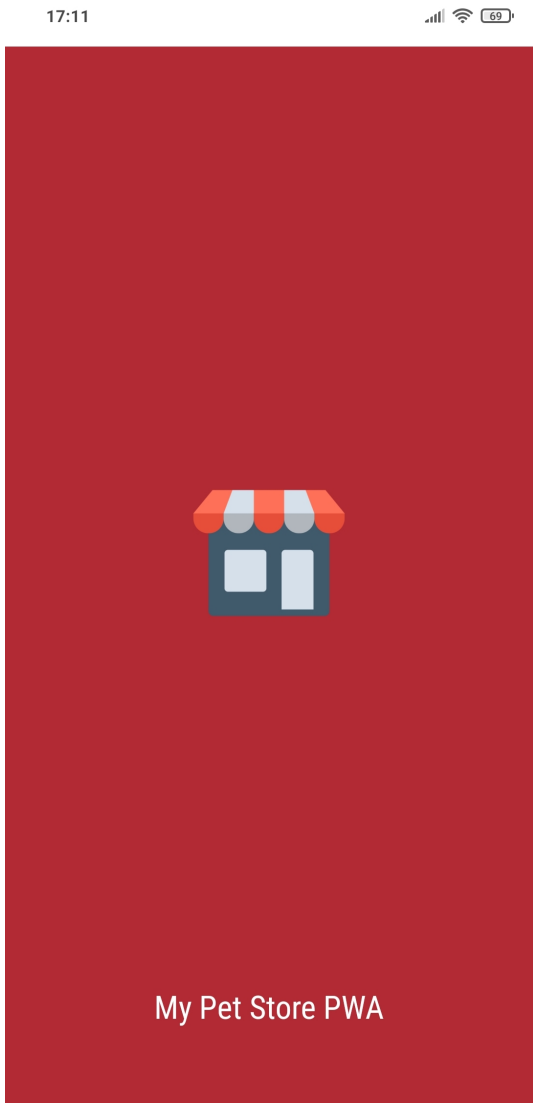
- A web manifest, with the correct fields filled in
- The website to be served from a secure (HTTPS) domain
    - We will use GitHub Pages, which provide HTTPS connection
- An icon to represent the app on the device
- A service worker registered, to make the app work offline

Now you can install it on your phone









Reading

MDN: Introduction to progressive web apps